Programmazione I e laboratorio: prova pratica
Tempo a disposizione: 2 ore

Corso B

17/01/2020

Implementare il programma richiesto usando il linguaggio standard ANSI C e sottomettere la soluzione sulla piattaforma di esame.

Il codice consegnato sarà valutato solo se supera almeno il 51% dei test case. La valutazione terrà conto della correttezza del programma, strutture dati usate, uso efficiente di memoria, efficienza dell'implementazione algoritmica, stile di programmazione, controlli dell'input.

Non è consentito l'uso di nessun materiale o strumento tecnologico oltre il computer del laboratorio. Si può usare qualsiasi IDE installato sul computer.

Non è consentita la collaborazione tra studenti.

Se si desidera abbandonare l'esame occorre dirlo esplicitamente ai docenti per evitare la consegna del codice.

Compilazione Si ricorda di compilare con l'opzione ANSI C e che per abilitare i messaggi di diagnostica del compilatore, bisogna compilare il codice usando le opzioni `-g -Wall` di gcc:

```
gcc -std=c89 -Wall -g sorgente.c -o eseguibile
```

Provare la propria soluzione in locale. Valutare la correttezza della soluzione sulla propria macchina accertandosi che rispetti gli input/output contenuti nel TestSet. I file di input e output per i test sono nominati secondo lo schema:
`input0.txt output0.txt`
`input1.txt output1.txt`
...
Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio
`./eseguibile < input0.txt`
effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che `eseguibile` sia il file ottenuto dalla compilazione del vostro codice. Per effettuare un controllo automatico sul primo file input `input0.txt` e trovare le differenze fra l'output prodotto dal vostro programma e quello corretto potete eseguire la seguente sequenza di comandi

```
./eseguibile < input0.txt | diff - output0.txt
```

Esercizio

Vengono lette da standard input due matrici. L'input è così organizzato:

- La prima matrice:
  - La prima riga contiene due interi N_1 e M_1, separati da spazio, che rappresentano il numero di righe e colonne della prima matrice.
  - Seguono N_1 righe, ogni una contenente M_1 interi separati da spazio, che rappresentano le righe della prima matrice.
- La seconda matrice in modo analogo:
  - La prima riga contiene due interi N_2 e M_2, separati da spazio, che rappresentano il numero di righe e colonne della seconda matrice.
  - Seguono N_2 righe, ogni una contenente M_2 interi separati da spazio, che rappresentano le righe della seconda matrice.

Si può assumere che l'input sia sempre corretto (non è necessario fare il controllo dell'input).

Verificare se gli elementi di una matrice sono tutti contenuti nell'altra. Sia A la matrice più piccola e B la matrice più grande. Se le matrici hanno lo stesso numero di elementi, allora si consideri A la prima matrice letta in input e B la seconda. Se un elemento compare nella matrice A n volte, deve comparire nella matrice B almeno n volte. In caso negativo stampare `NO`.

In caso positivo, verificare se la matrice A verifica la proprietà :

$$\exists j \in [0, M).(\exists i \in [0, N).(\forall k \in [i, N).A[k, j] < 0)) \tag{1}$$

dove N e M sono il numero di righe e colonne di A. Stampare la matrice formata dalle colonne j di A che verificano la condizione precedente. Ognuna di tali colonne deve essere stampata per indice di riga decrescente. Nell'output gli elementi delle righe sono separati da `';'`.

Si ricorda che non vi deve essere spreco nell'uso della memoria. Sarà valutata anche l'organizzazione del codice: uso delle funzioni, commenti, indentazione, semplicità.

Esempio

Input

```
2 3
1 2 3
4 -5 -6
3 4
1 2 3 8
4 -12 -6 -7
9 0 -1 -5
```

Output

```
-5;-6
2;3
```

Input

```
2 3
1 2 3
4 5 6
3 4
1 2 3 8
4 -12 -6 -7
9 0 -1 -5
```

Output

```
NO
```

UNIVERSITÀ DI PISA

PROGRAMMAZIONE 1
Corso B

documentazione per l'esame

corrado priami - corrado.priami@unipi.it

convertitore	descrizione
d	stampa intero decimale con segno
i	stessa cosa di d — sarà diverso per scanf
o	stampa intero senza segno in ottale
u	stampa intero decimale senza segno
x oppure X	stampa intero decimale senza segno in esadecimale (x usa a-f, X usa A-F come cifre)
h, l oppure ll	prima di qualsiasi convertitore intero per short, long e long long (length modifiers)
e oppure E	stampa floating point in notazione esponenziale
f	stampa floating point in notazione fixed point
g oppure G	come f o e (E) a seconda della grandezza del numero
L	prima di qualsiasi convertitore floating-point per indicare long double
c	stampa caratteri
s	stampa stringhe
p	stampa l'indirizzo delle variabili
%	stampa %

	precisione
interi	numero minimo di cifre da stampare (default 1). Se maggiore delle cifre dell'intero si estendono gli eventuali 0 iniziali, 0 dopo la virgola oppure si premettono spazi per giustificare a destra
reali	con e, E, f è il numero di cifre dopo la virgola; con g e G é il numero di cifre significative da stampare
stringhe	il numero massimo di caratteri da stampare dall'inizio della stringa

printf

flag	descrizione
-	giustifica a sinistra l'output nel campo corrispondente
+	stampa il segno + (-) davanti ai numeri positivi (negativi)
spazio	stampa uno spazio davanti ai valori positivi al posto del + se non stampati con flag +
#	prefigge 0 ad output con %o (ottale), prefigge 0x (0X) ad output con %x (%X), forza la stampa del punto decimale negli output con %e, %E, %g, %G, %f anche in assenza di parte decimale
0	tiempo un campo con 0 iniziali davanti al dato stampato

escape set	descrizione
\'	stampa il singolo apice
\"	stampa apice doppio
\?	stampa punto interrogativo
\\	stampa il backslash
\a	genera un alert sonoro o visuale
\b	muove il cursore indietro di una posizione sulla linea corrente
\f	muove il cursore all'inizio della successiva pagina logica
\n	muove il cursore all'inizio della linea successiva
\r	muove il cursore all'inizio della linea corrente
\t	muove il cursore alla posizione del prossimo tab orizzontale
\v	muove il cursore alla posizione del prossimo tab verticale

convertitore	descrizione
d	legge intero decimale opzionalmente con segno
i	legge intero decimale opzionalmente con segno, ottale o esadecimale
o	legge intero ottale
u	legge intero decimale senza segno
x oppure X	legge intero esadecimale (x usa a-f, X usa A-F come cifre)
h, l oppure ll	prima di qualsiasi convertitore intero per short, long e long long (length modifiers)
e, E, f, g o G	legge valori floating point
l oppure L	prima di qualsiasi convertitore floating-point per indicare double o long double
c	legge caratteri
s	legge stringhe
[char set]	legge input finché trova caratteri contenuti nel set
n	memorizza il numero di caratteri letti fino a quel momento
p	legge un indirizzo
%	salta i caratteri % nell'input

`<math.h>`

funzione	descrizione
sqrt(x)	radice quadrata
cbrt(x)	radice cubica
exp(x)	funzione esponenziale
log(x)	logaritmo naturale di x
log10(x)	logaritmo in base 10 di x
fabs(x)	valore assoluto di x come floating number
ceil(x)	arrotonda al più piccolo intero non minore di x
floor(x)	arrotonda al più grande intero non maggiore di x
pow(x, y)	x elevato alla y
fmod(x, y)	resto di x/y come floating number
sin(x)	seno di x in radianti
cos(x)	coseno di x in radianti
tan(x)	tangente di x radianti

file accesso casuale

i file ad accesso casuale sono organizzati in slot di dimensione fissata e consentono di accedervi senza dover scorrere tutto il file — si può calcolare la posizione di un record come offset dall'inizio del file e di una chiave di accesso al record

è possibile inserire nuovi dati senza distruggere quelli già presenti
è possibile leggere i dati sia sequenzialmente che casualmente
è possibile aggiornare i dati
è possibile cancellare record

è possibile posizionare il puntatore del file ad una posizione specifica con

```
int fseek(FILE *, long, int)
```

dove il secondo parametro è l'offset (distanza) in byte da un punto del file dove cominciare a scrivere individuato dal terzo parametro che è una costante simbolica tra SEEK_SET (offset dall'inizio del file), SEEK_CUR (offset dalla posizione corrente) e SEEK_END (offset dalla fine del file) — ritorna un valore non nullo

```
int fprintf(FILE *, const char *, …)
int fscanf(FILE *, const char *, …)
```

scrivono e leggono il numero di caratteri necessario a rappresentare testualmente il dato (es. 1 si scrive un carattere)

```
int fwrite(const void *, size_t, size_t, FILE *)
int fread(const void *, size_t, size_t, FILE *)
```

scrivono e leggono array su e da file per un numero di byte passato come secondo argomento a partire da una posizione iniziale indicata dal puntatore al file. Il terzo argomento rappresenta il numero di elementi nell'array — variabili singole sono trattate come array di un solo elemento

ritornano tutte il numero di scritture o letture effettuate con successo

`<stdio.h>`

Prototipo	Descrizione
`int getchar(void);`	legge il carattere successivo dallo standard input e lo restituisce come intero
`int putchar(int c);`	stampa il carattere memorizzato in `c` e lo restituisce come intero
`int puts(const char *s);`	stampa una stringa costante passata come argomento seguita da newline; ritorna un intero `> 0` se eseguita con successo ed `EOF` altrimenti
`int sprintf(char *s, ..);`	il primo parametro formale è un array di caratteri, i rimanenti parametri sono gli stessi di `printf`; anziché stampare a schermo, memorizza l'output nell'array `s` — ritorna il numero di caratteri scritti in `s`, oppure `EOF` se si verifica un errore
`int sscanf(char *s, ..);`	il primo parametro formale è un array di caratteri, i rimanenti parametri sono gli stessi di `scanf`; anziché leggere da tastiera, legge dall'array `s` — ritorna il numero di caratteri letti con successo, oppure `EOF` se si verifica un errore
`char *fgets(char *s, int n, FILE *stream);`	legge caratteri dallo `stream` specificato nel terzo parametro e li memorizza nell'array di caratteri `s` fino a che non incontra un newline oppure ha letto `n-1` byte (cioè `n-1` caratteri); il carattere `NULL` è appeso al termine dell'array — lo stream `stdin` è lo standard input, tipicamente la tastiera; ritorna la stringa letta e memorizzata in `s` — se incontra un newline, viene incluso nella stringa letta

<ctype.h>

Prototipo	Descrizione
`int isblank(int c);`	se **c** è uno spazio che separa parole in una linea di testo ritorna vero, altrimenti falso (**0**)
`int isdigit(int c);`	se **c** è una cifra ritorna vero, altrimenti falso (**0**)
`int isalpha(int c);`	se **c** è una lettera ritorna vero, altrimenti falso (**0**)
`int isalnum(int c);`	se **c** è una cifra o una lettera ritorna vero, altrimenti falso (**0**)
`int isxdigit(int c);`	se **c** è una cifra esadecimale ritorna vero, altrimenti falso (**0**)
`int islower(int c);`	se **c** è una lettera minuscola ritorna vero, altrimenti falso (**0**)
`int isupper(int c);`	se **c** è una lettera maiuscola ritorna vero, altrimenti falso (**0**)
`int tolower(int c);`	se **c** è una lettera maiuscola ritorna **c** minuscola, altrimenti **c** senza cambiamenti
`int toupper(int c);`	se **c** è una lettera minuscola ritorna **c** maiuscola, altrimenti **c** senza cambiamenti
`int isspace(int c);`	se **c** è un carattere separatore spaziale (newline '\n', spazio ' ', form feed '\f', tab orizzontale '\t', tab verticale '\v', return '\r') ritorna vero, altrimenti falso (**0**)
`int iscntrl(int c);`	se **c** è un carattere di controllo (newline '\n', form feed '\f', alert '\a', backspace '\b', tab orizzontale '\t', tab verticale '\v', return '\r') ritorna vero, altrimenti falso (**0**)
`int ispunct(int c);`	se **c** è un carattere di stampa (cioè visibile a video) diverso da cifre, lettere e spazio (es. **$,#, (,), [,], {, }, ;, :, %**, ecc.) ritorna vero, altrimenti falso (**0**)
`int isprint(int c);`	se **c** è un carattere di stampa incluso spazio ritorna vero, altrimenti falso (**0**)
`int isgraph(int c);`	se **c** è un carattere di stampa escluso spazio ritorna vero, altrimenti falso (**0**)

`<string.h>`

Prototipo	Descrizione
`char *strcpy(char *s1, const char *s2);`	copia la stringa `s2` nell'array `s1` e ritorna il valore di `s1`
`char *strncpy(char *s1, const char *s2, size_t n);`	copia al più `n` caratteri della stringa `s2` nell'array `s1` e ritorna il valore di `s1`
`char *strcat(char *s1, const char *s2);`	appende la stringa `s2` all'array `s1`; il primo carattere di `s2` sovrascrive il carattere terminale `NULL` di `s1` e ritorna il valore di `s1`
`char *strncat(char *s1, const char *s2, size_t n);`	appende al più `n` caratteri della stringa `s2` all'array `s1`; il primo carattere di `s2` sovrascrive il carattere terminale `NULL` di `s1` e ritorna il valore di `s1`
`int strcmp(const char *s1, const char *s2);`	confronta `s1` e `s2` e ritorna `0`, `>0` o `<0` se `s1` è uguale, maggiore o minore di `s2` secondo l'ordinamento lessicografico
`int strncmp(const char *s1, const char *s2, size_t n);`	confronta fino a `n` caratteri della srtinga `s1` con la stringa `s2` e ritorna `0`, `>0` o `<0` se `s1` è uguale, maggiore o minore di `s2` secondo l'ordinamento lessicografico
`char *strchr(const char *s, int c);`	identifica la posizione del carattere `c` nella stringa `s` e restituisce un puntatore a `c`, altrimenti il puntatore `NULL`
`size_t strcspn(const char *s1, const char *s2);`	ritorna la lunghezza del primo segmento di `s1` formato solo da caratteri non contenuti in `s2`
`size_t strspn(const char *s1, const char *s2);`	ritorna la lunghezza del primo segmento di `s1` formato solo da caratteri contenuti in `s2`
`char *strpbrk(const char *s1, const char *s2);`	identifica la prima posizione in `s1` che contiene un carattere di `s2` e ritorna un puntatore a tale carattere, altrimenti lil puntatore `NULL`
`char *strrchr(const char *s1, int c);`	identifica l'ultima occorrenza di `c` nella stringa `s` e ritorna un puntatore a `c`, altrimenti ritorna lil puntatore `NULL`
`char *strstr(const char *s1, const char *s2);`	localizza la prima occorrenza di `s2` in s1 e ritorna un puntatore a dove `s2` inizia in `s1`, altrimenti il puntatore `NULL`
`char *strtok(char *s1, const char *s2);`	una serie di chiamate divide `s1` in token separati da caratteri contenuti in `s2` — le chiamate successive alla prima hanno `NULL` come primo argomento; ciascuna chiamata ritorna un puntatore al token identificato e quando non ve ne sono più ritorna il puntatore `NULL`
`size_t strlen(const char *s);`	ritorna il numero di caratteri in `s` escluso il sterminatore `NULL`

`<string.h>`

Prototipo	Descrizione
`void *memcpy(void *s1, const void *s2, size_t n);`	copia `n byte` dell'oggetto puntato da `s2` nell'oggetto puntato da `s1` e restituisce un puntatore all'oggetto risultante — indefinita se gli oggetti puntati da `s1` e `s2` sono anche parzialmente sovrapposti in memoria
`void *memmove(void *s1, const void *s2, size_t n);`	copia `n byte` dell'oggetto puntato da `s2` nell'oggetto puntato da `s1` e restituisce un puntatore all'oggetto risultante — la copia è usata facendo uso di un temporaneo in modo da poter copiare una parte di una stringa in un'altra parte della stessa stringa
`int memcmp(const void *s1, const void *s2, size_t n);`	confronta i primi `n byte` degli oggetti puntati da `s1` e `s2` — ritorna `0` se `s1 == s2`, `<0` se `s1 < s2` e `>0` se `s1 > s2`
`void *memchr(const void *s, int c, size_t n);`	ritorna un puntatore alla prima occorrenza di `c` (interpretato come `unsigned char`) in `s` oppure `NULL` se non trovata
`void *memset(void *s, int c, size_t n);`	copia `c` (interpretato come `unsigned char`) nei primi `n byte` dell'oggetto puntato da `s` e ritorna un puntatore al risultato
`char *strerror(int errornum)`	traduce i codici errore in frasi partendo dall'header `<errno.h>` e restituisce un puntatore a una stringa

operatore	Descrizione
&	AND — confronta due operandi bit a bit e restituisce AND dei singoli bit
|	OR — confronta due operandi bit a bit e restituisce OR dei singoli bit
^	XOR — confronta due operandi bit a bit e restituisce XOR dei singoli bit (vero solo se i bit sono diversi)
<<	sposta i bit del primo operando a sinistra del numero di posizioni indicate dal secondo operando e riempe le posizioni lasciate vuote con 0
>>	sposta i bit del primo operando a destra del numero di posizioni indicate dal secondo operando e riempe le posizioni lasciate vuote con 0
~	complementa (`1` in `0` e `0` in `1`) tutti i bit del singolo operando
&=	`x &= y` è equivalente a `x = x & y`
|=	`x |= y` è equivalente a `x = x | y`
^=	`x ^= y` è equivalente a `x = x ^ y`
<<=	`x <<= y` è equivalente a `x = x << y`
>>=	`x >>= y` è equivalente a `x = x >> y`

modo	descrizione	modo	descrizione
`r`	apre un file esistente in lettura	`rb`	come `r`, ma file binario
`w`	crea un file in scrittura — se già esiste ne cancella il contenuto e lo apre in scrittura	`wb`	come `w`, ma file binario
`a`	apre un file per appenderne (scrittura) contenuto alla fine	`ab`	come `a`, ma file binario
`r+`	apre un file esistente per modifica (lettura e scrittura)	`rb+`	come `r+`, ma file binario
`w+`	crea un file in lettura e scrittura — se già esiste ne cancella il contenuto e lo apre in scrittura	`wb+`	come `w+`, ma file binario
`a+`	apre un file per modifica (lettura e scrittura), ma scritture sono fatte solo alla fine del file	`ab+`	come `a+`, ma file binario

`<starg.h>`

Identificatore	Descrizione
`va_list`	è un tipo che rappresenta la lista di argomenti in modo adeguato per l'uso delle macro `va_start`, `va_arg` e `va_end` — un oggetto di tipo `va_list` deve essere definito nella funzione e usato per accedere correttamente agli argomenti
`va_start(va_list, int)`	è una macro da invocare prima di poter accedere agli argomenti di una lista di parametri a lunghezza variabile — inizializza gli elementi dichiarati con `va_list` per renderli usabili da `va_start` e `va_end` — l'intero rappresenta il numero di parametri in `va_list`
`va_arg(va_list, type)`	è una macro che accede all'elemento corrente della lista di argomenti del tipo specificato nel suo secondo parametro e sposta il puntatore in `va_list` al prossimo argomento
`va_end(va_list)`	è una macro che ripulisce la lista di argomenti a lunghezza variabile per consentire alla funzione di ritornare correttamente (gestisce opportunamente il record di attivazione)

UNIVERSITÀ
DI PISA

PROGRAMMAZIONE 1
Corso B

in bocca al lupo!!!!

corrado priami - corrado.priami@unipi.it